© Hans-Georg Beckmann 2002

Technische Informatik - Hardware

Teil 1: Grundlagen

Vorbemerkungen	2
Dezimalzahlen, Dualzahlen, Hexzahlen	3
Umrechnen in Zahlensystemen	4
Addieren zweier Dualzahlen	6
Halbaddierer	7
Logikbausteine, Schaltalgebra	8
Halbaddierer bauen	10
TTL-Bausteine, Material	11
Schaltungsentwicklung, Volladdierer	13
2 x 4Bit-Volladdierer	16
Übungsaufgaben	18
.....Demultiplexer und Multiplexer	18
.....Schwellwertschaltung	20
.....Siebensegmentanzeige	20
.....BCD-Zahlen	21
.....Vereinfachen von Logikgleichungen	22
.....UND-Form der Logikgleichung	23
.....weitere Übungen	26

Teil 2: Speicherbausteine	27
Speichern mit Flip - Flops	27
Zeitllaufdiagramme für Flip-Flops	30
Eine einfache Anwendung	31
Verbesserung der Flip-Flops	32
.....RS-FF mit dominierenem Eingang	32
.....RS-FF mit Zustandsteuerung	34
D-Flip-Flops und Verbesserungen	36
J-K - Flip - Flops	38
Das universelle J-K-MS-Flip-Flop	39
Frequenzen teilen	41
.....JK-MS-FF als T-Flip-Flop	43
.....JK-MS-FF als Zähler	44
.....weitere Frequenzteiler	45
.....BCD-Zähler	50
Aufgaben	50
Daten sc hieben	51
Rechtsschieben, Linksschieben	53
Universalregister	55
Adressierbare Speicher	61

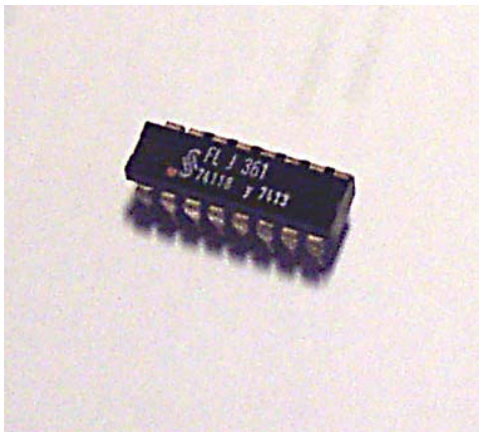
Literatur: Zusätzlich zu dem VLIN - Hardwarescript sind folgende Bücher sinnvoll, wenn sie denn noch zu haben sind:

Einführung in die Digitalelektronik- Band 1 - 3, Herausgeber Jean Pütz
Verlagsgesellschaft Schulfernsehen - vgs- Köln 1983
TTL Taschenbuch Teil 1 und 2, Verlag iwt München, 1987

© Hans-Georg Beckmann 2002

Technische Informatik - Hardware

Vorbemerkungen



Die Behandlung digitaler Schaltungen und technischer Grundlagen der Informatik sollte in jedem Falle Teil des Informatikunterrichts in der Sekundarstufe 2 sein. In den RRL finden sich entsprechende Hinweise unter 2.2, wenn es heißt: *" Das logisch-technische Konzept, das dem Wechselspiel zwischen Hard - und Software zugrunde liegt, ist exemplarisch zu erarbeiten."*

Und weiter werden als Stichpunkte angegeben.

- Codierung von Zahlen und Zeichen

- Funktionsprinzipien wesentlicher Systemkomponenten

- Entwicklung und Realisierung von Schaltnetzen durch Elektronikbausteine oder Programmsimulation

Die Behandlung von Schaltwerken kann dabei nicht isoliert behandelt werden. Vielmehr bietet sich an, Inhalte der theoretischen Informatik - hier die Automatehntheorie - mit zu behandeln, da diese Aspekte eng miteinander verbunden sind.

Es kann bei der Behandlung der "harten Ware" nicht darum gehen, im Unterricht das Blockdiagramm des Pentium IV zu besprechen und über die Rolle des Level3 Cache zu beraten , nur um aktuell und interessant zu sein [1]. Es geht um Grundlagen, die im wahrsten Sinne des Wortes für Schülerinnen und Schüler zu begreifen sind. Die Behandlung der technischen Informatik bietet auch die Möglichkeit, "handwerklich" etwas im Informatikunterricht zu tun, was ja sonst in unserem Fach nicht so oft vorkommt.

[1] Oft ist man verführt diesen aktuellen technischen Entwicklungen einen großen Raum zu geben. Da aber in Wahrheit weder Schülerinnen und Schüler noch Lehrerinnen und Lehrer die notwendigen Detailkenntnisse haben, bleiben solche Betrachtungen zwangsweise oberflächlich. Denken sie an den Physikunterricht. Dort käme niemand auf den Gedanken, bei der Behandlung von Verbrennungsmotoren die neue Mercedes S-Klasse genauer zu bearbeiten. Warum also sollten wir im Informatikunterricht auf solche aktuellen technischen Entwicklungen Rücksicht nehmen ? Was im Unterricht zählt, sind zuerst einmal Grundlagenkenntnisse.

© Hans-Georg Beckmann 2002

Dualzahlen, binäre Zahlendarstellung, Hexadezimalzahlen

Aus der Sekundarstufe 1 sind die Dualzahlen möglicherweise bekannt. Eine kleine Tabelle stellt die Zahlensysteme, mit denen wir es zu tun bekommen vor:

Dezimal	Dual	Hexadezimal
0	00000	00
1	00001	01
2	00010	02
3	00011	03
4	00100	04
5	00101	05
6	00110	06
7	00111	07
8	01000	08
9	01001	09
10	01010	0A
11	01011	0B
12	01100	0C
13	01101	0D
14	01110	0E
15	01111	0F
16	10000	10
17	10001	11
...	....	....

Dabei basieren die Dualzahlen auf Zweierpotenzen und berechnen sich wie folgt:

	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Dezimal 13 =	**0**	**1**	**1**	**0**	**1**
	2^4	2^3	2^2	2^1	2^0

Dezimal 13 = $\mathbf{0} \times 2^4 + \mathbf{1} \times 2^3 + \mathbf{1} \times 2^2 + \mathbf{0} \times 2^1 + \mathbf{1} \times 2^0 =$

$= 0 + 8 + 4 + 0 + 1 = 13$

Die einzelnen Stellen der Dualzahl heißen Bits. 4 Bit heißen 1 Nibble , 8 Bit heißen 1 Byte und 16 Bit heißen 1 Word.

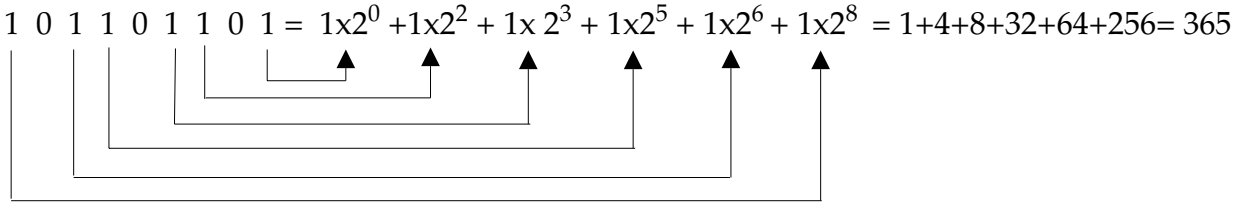
Mit n Stellen kann man in einer Dualzahlen alle Dezimalzahlen von 0 bis 2^n -1 darstellen.

Mit einem Byte lassen sich damit Zahlen von 0 bis 255 (2^8 -1) darstellen. Mit einem word lassen sich Zahlen von 0 bis 65535 (2^{16} -1) darstellen.

© Hans-Georg Beckmann 2002

Umrechnung von Dualzahlen in Dezimalzahlen

Man beginnt rechts in der Dualzahl mit der Poten 2^0 und arbeitet sich nach links bis an den Anfang der Zahl vor, wobei mit jeder Stelle der Exponent um 1 zunimmt.



Die Bits, die eine Null enthalten spielen bei der Summenbildung keine Rolle, da die entsprechende Potenz mit Null multipliziert wird.

Umrechnung von Dezimalzahlen in Dualzahlen

Man spaltet zuerst die größtmögliche Zweierpotenz ab und probiert dann vom Rest die nächst kleinere Potenz abzuspalten. Das macht man mit dem nachfolgenden Rest wieder, bis die Zahl passend zerlegt ist: Beispiel : 23567 = 101110000001111

23567	=	1	x	16384	+	7183	=	**1**	x	2^{14}	+	7183
7183	=	0	x	8192	+	7183	=	**0**	x	2^{13}	+	7183
7183	=	1	x	4096	+	3087	=	**1**	x	2^{12}	+	3087
3087	=	1	x	2048	+	1039	=	**1**	x	2^{11}	+	1039
1039	=	1	x	1024	+	15	=	**1**	x	2^{10}	+	15
15	=	0	x	512	+	15	=	**0**	x	2^{9}	+	15
15	=	0	x	265	+	15	=	**0**	x	2^{8}	+	15
15	=	0	x	128	+	15	=	**0**	x	2^{7}	+	15
15	=	0	x	64	+	15	=	**0**	x	2^{6}	+	15
15	=	0	x	32	+	15	=	**0**		2^{5}	+	15
15	=	0	x	16	+	15	=	**0**		2^{4}	+	15
15	=	1	x	8	+	7	=	**1**		2^{3}	+	7
7	=	1	x	4	+	3	=	**1**		2^{2}	+	3
3	=	1	x	2	+	1	=	**1**		2^{1}	+	1
1	=	1	x	1	+	0	=	**1**		2^{0}	+	0

© Hans-Georg Beckmann 2002

Hexadezimalzahlen

Diese sind offensichtlich erfunden worden, weil man sich ellenlange Dualzahlen nicht merken konnte und beim Aufschreiben immer soviel Papier verbraucht wurde. Hier ist die Basis die Zahl 16. In der kleinsten Stelle hat man 16^0, dann 16^1, dann 16^2 usw.
Das bedeutet aber auch, dass ,man pro Stelle 16 symbole (Ziffern) haben muss. Zu Verweirrung aller Schülerinnen und Schüler hat man hier erst eimal die Ziffern des Zehnersystems benutzt (0 bis 9) und dann mit dem Alphabeth weitergemacht (A bis F siehe auch Tabelle auf Seite 1).
Man könnte meinen, dass es doch viel übersichtlicher gewesen wäre, neuen Symbole zu erfinden, um Missverständisse zu vermeiden. Das ist ein guter Gedanke aber im Zeitalter der Schreibmaschine, war es mit neuen Symbolen nicht so weit her. Damit man nun unterscheiden kann, ob zB. mit " 12" die Zahl Zwölf aus dem Dezimalsystem gemeint ist oder die 12 aus dem Hexadezimalsystem (was der Dezimalzahl 18 entspricht) hat man sich angewöhnt , vor die Hexadezimalzahlen ein Dollarzeichen zu setzen. Damit wäre also: $12 = 18

Umrechnung Hexadezimalzahlen in Dezimalzahlen

Will man beispielsweise die Hexadezimalzahl $FF37 in eine Dezimalzahl umrechnen, dann verfährt man analog zu den Dualzahlen. Die Stelle ganz rechts entspricht 16^0 und es geht nach links weiter bis zur Stelle, die 16^3 entspricht:

$$\$FF37 = \mathbf{F} \times 16^3 + \mathbf{F} \times 16^2 + \mathbf{3} \times 16^1 + \mathbf{7} \times 16^0 = \mathbf{15} \times 16^3 + \mathbf{15} \times 16^2 + \mathbf{3} \times 16^1 + \mathbf{7} \times 16^0 =$$
$$= \mathbf{15} \times 4096 + \mathbf{15} \times 256 + \mathbf{3} \times 16 + \mathbf{7} = 65335$$

Es soll Leute geben, die das im Kopf berechnen. Außerdem gibt es Taschenrechner, die das können.
Es wird aber schon deutlich, dass in jedem Falle Zahlen entstehen, die weit übersichtlicher sind als Dualzahlen. Als PCs noch klein und freundlich waren, hatten sie 64 KB Speicher. Um genau zu sein: Es gab 65536 Speicherstellen zu je ein Byte. Die waren durchgezählt, was von Speicherstelle 0000000000000000 bis 1111111111111111 ging. Da sieht es doch deutlich besser aus, wenn es von $0000 bis $FFFF geht.

Auf weitere Umrechnungen sei hier verzichtet.

© Hans-Georg Beckmann 2002

Rechnen mit Dualzahlen

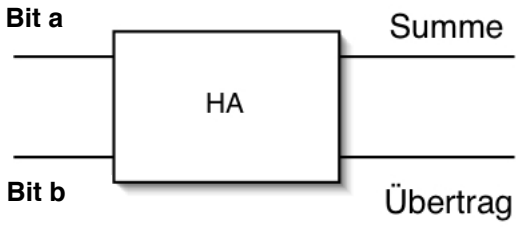
Wahrscheinlich ist es nur eine Wiederholung aus der Sekundarstufe 1, aber schaden kann es nichts, wenn wir versuchen, zwei Dualzahlen zu addieren. Wie auch im Zehnersystem werden die einzelenen Stellen addiert. Wenn dabei ein Ergebnis herauskommt, dass den Bereich der für jede Stelle zulässigen Ziffern überschreitet, gibt es einen Übertrag in die Position links daneben. Dabei wird sicher gelten: 0 + 0 = 0 und 1 + 0 = 1 und 0 + 1 = 1 und 1 + 1 = 0 mit Übertrag 1. Gerade der letzte Punkt macht Schülerinnen und Schülern anfangs etwas Probleme. Es wird aber schnell klar, dass ja 1 + 1 = 2 ist, aber die Zahl 2 in der Dualschreibeweise ja "10" ist und somit die Rechnung stimmt.

Beispiel:

	7	--->	0	1	1	1
+	6	--->	0	1	1	0
		Übertrag	1	1		
	13	<---------	1	1	0	1

Nun soll eine Maschine gebaut werden, die vorerst einmal ganz rechts in dieser Rechnung die Addition der Dualzahlen korrekt durchführen kann.

Die Maschine heißt *Halbaddierer* und sieht als Blockbild genauso einfach aus, wie die Tabelle, die die Funktion beschreibt:



Bit a	Bit b	Summe	Übertrag
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Nun brauchen wir nur noch die elektrischen Bauteile, mit denen wir Nullen und Einsen realisieren können. Null kann "Schalter offen" oder "Strom aus" oder "Licht aus" oder "Spannung 0 V " bedeuten, während dann "Schalter geschlossen" oder "Licht an" oder "Strom an " oder "Spannung 5V" der Eins entspricht.

Mit einfachen Schaltern wäre im Prinzip die obige Tabelle dann wie folgt zu Übersetzen:

Die Summenlampe leuchtet, wenn (Schalter a *nicht* geschlossen *und* Schalter b geschlossen) *oder* (Schalter a geschlossen *und* Schalter b *nicht* geschlossen) sind.

Die Übertragslampe leuchtet, wenn Schalter a geschlossen *und* Schalter b geschlossen sind.

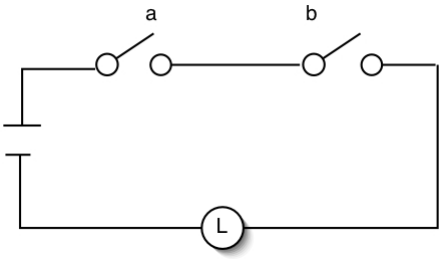
© Hans-Georg Beckmann 2002

Das kann man mit den Symbolen der Schaltalgebra auch kürzer haben:

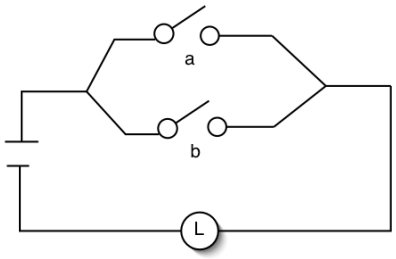
$$\text{Summe} = (\bar{a} \wedge b) \vee (a \wedge \bar{b})$$

$$\text{Übertrag} = a \wedge b$$

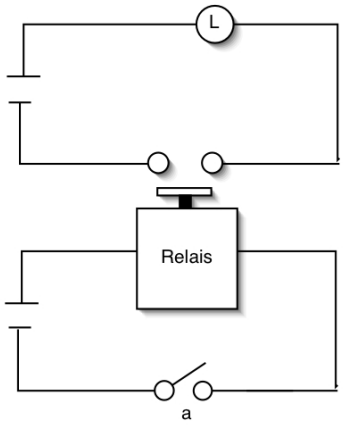
Dabei ist $\wedge$ das Zeichen für das logische UND. Weiterhin ist $\vee$ das Zeichen für das logische ODER und der Querstrich über einem Symbol bedeutet NICHT.
Diese logischen Verknüpfungen könnten im Physikraum leicht gebaut werden:



Die Lampe **L** leuchtet ,wenn a *und* b geschlossen sind

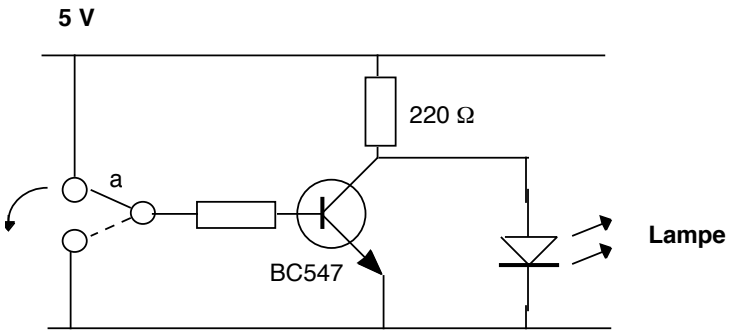


Die Lampe **L** leuchtet ,wenn a *oder* b geschlossen ist (auch beide)



Ein Relais sei so geschaltet, dass es im oberen Stromkreis unterbricht , wenn unten der Stromkreis mit Schalter a geschlossen wird. L leuchtet, wenn *nicht* a geschlossen ist.

Besser geht das natürlich mit Transistoren. In der nebenstehenden Schaltung schaltet der Transistor durch, wenn **a** auf 1 liegt. Dann leuchtet die Lampe nicht mehr. Sperrt der Transistor, weil **a** auf 0 V liegt, dann fließt Strom durch die Lampe, sie leuchtet. Die Schaltung entspricht einem NICHT **a**.



Das soll als kurzer Blick in das Innenleben logischer Schaltungen reichen. Wir werden fertige Bausteine benutzen (zuerst sind die aus Kreide und leben nur an der Tafel oder auf dem Blatt Papier später werden die dann auch reale Objekte).

© Hans-Georg Beckmann 2002

Die grundlegenden Bausteine

Die Schaltungen, die wir aufbauen wollen, basieren auf wenigen Grundbausteinen (Gattern), deren Funktion, Logiktabelle und Symbol im Folgenden aufgelistet sind.

Die Eingänge der Bausteine sind jeweils mit **a** oder **b** bezeichnet. Der Ausgang heißt jeweils Q.

In der Logiktabelle werden immer alle möglichen Kombinationen der Eingangswerte a und b (oder auch nur a) aufgelistet und es wird immer angegeben, was dann der Ausgang Q macht.

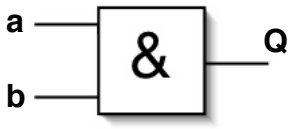
Schaltsymbol DIN

a	b	Q
0	0	0
0	1	0
1	0	0
1	1	1

Der UND-Baustein

Der Ausgang Q steht auf 1, wenn

$a \wedge b$

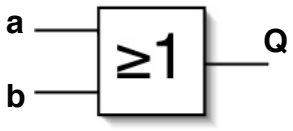


a	b	Q
0	0	0
0	1	1
1	0	1
1	1	1

Der ODER-Baustein

Der Ausgang Q steht auf 1, wenn

$a \vee b$

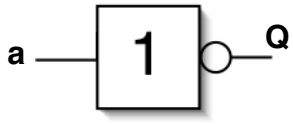


a	Q
0	1
1	0

Der NICHT-Baustein

Der Ausgang Q steht auf 1, wenn

$\overline{a}$



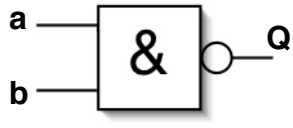
Neben diesen drei Bausteinen werden weiterhin gebraucht:

a	b	Q
0	0	1
0	1	1
1	0	1
1	1	0

Der NAND -Baustein

Der Ausgang Q steht auf 1, wenn nicht a *und* b also :

$\overline{a \wedge b}$

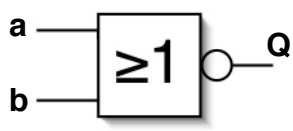


a	b	Q
0	0	1
0	1	0
1	0	0
1	1	0

Der NOR -Baustein

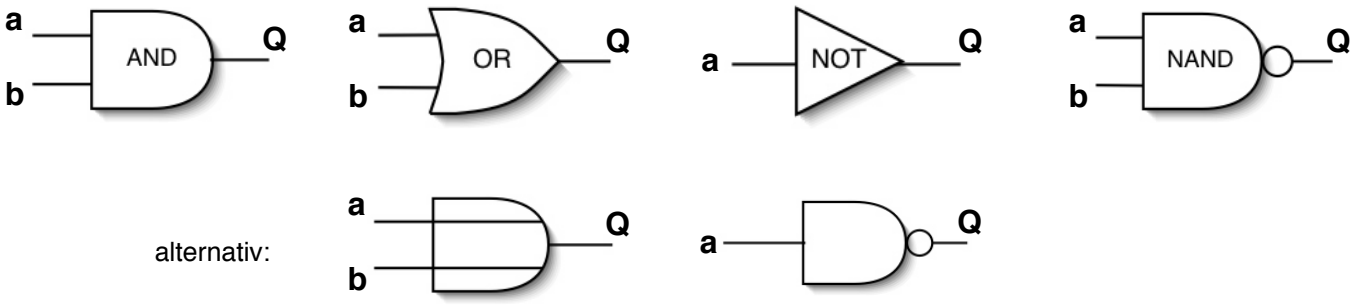
Der Ausgang Q steht auf 1, wenn nicht a *oder* b also :

$\overline{a \vee b}$



© Hans-Georg Beckmann 2002

In sehr vielen Büchern findet man immer noch die amerikanischen Symbole. Die sind weit weniger einprägsam, als die DIN-Symbole und sie werden hier nur der Vollständigkeit halber aufgeführt. Im Unterricht sollte man nur die DIN-Symbole verwenden.



Weiterhin werden wir noch einige Gesetze der Schaltalgebra brauchen.
Die wichtigsten sindneben dem Kommutativgesetz:

Assoziativgesetzt: $a \wedge (b \wedge c) = (a \wedge b) \wedge c = a \wedge b \wedge c$

$a \vee (b \vee c) = (a \vee b) \vee c = a \vee b \vee c$

Distributivgesetze: $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$

$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

Gesetze von de Morgan: $\overline{a \vee b} = \overline{a} \wedge \overline{b}$ und $\overline{a \wedge b} = \overline{a} \vee \overline{b}$

weitere Gesetze: $\overline{\overline{a}} = a$ (nicht (nicht a) = a)

$a \wedge 1 = a$

$a \vee 1 = 1$

$a \wedge 0 = 0$

$a \vee 0 = a$

Diese Gesetze werden immer wieder auftauchen und man sollte sie daher parat haben. Dass sie stimmen, kann man sich schnell mit einigen Tabellen klar machen. Ein Beispiel mag hier reichen:

a	b	$\overline{a}$	$\overline{b}$	$a \vee b$	$\overline{a \vee b}$	$\overline{a} \wedge \overline{b}$
0	0	1	1	0	1	1
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	1	0	0

Eines der deMorganschen Gesetzte tabellarsich dargestellt.

$$\overline{a \vee b} = \overline{a} \wedge \overline{b}$$

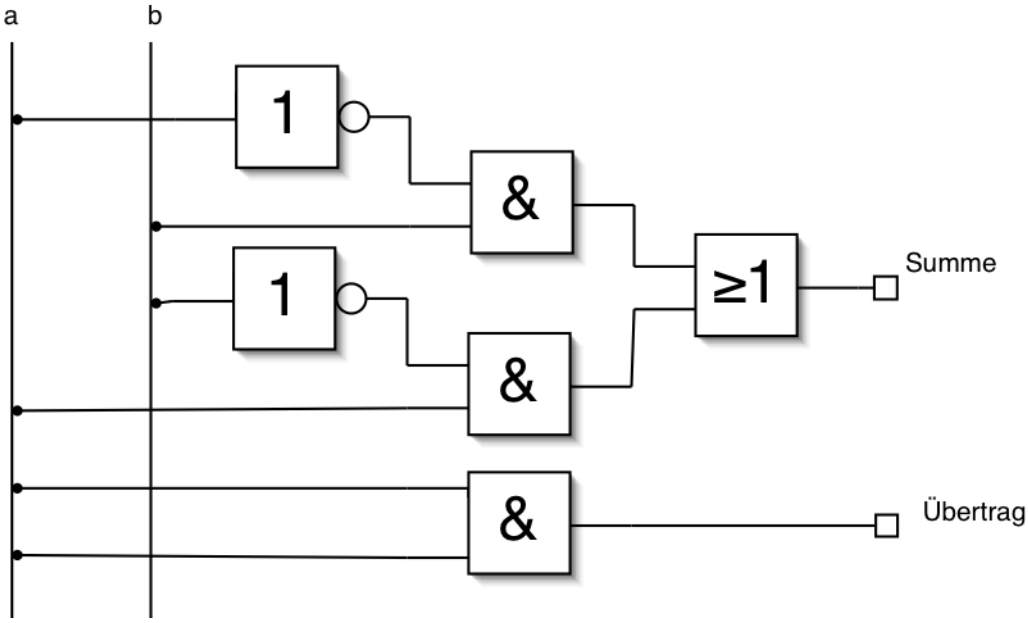
© Hans-Georg Beckmann 2002

Nun haben wir alle Teile zusammen, um den Halbaddierer zu bauen.

Die Logikgleichungen waren: $\text{Summe} = (\bar{a} \wedge b) \vee (a \wedge \bar{b})$

$\text{Übertrag} = a \wedge b$

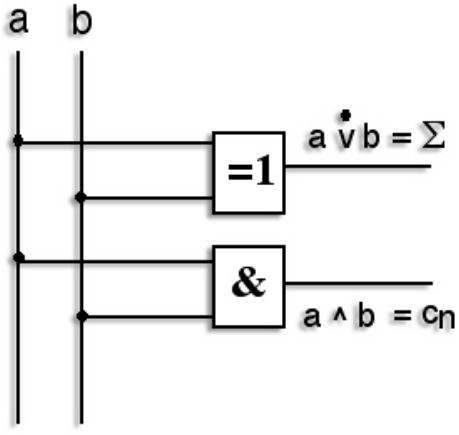
Die Schaltung wird den Gleichungen entsprechend zusammengebaut. Dabei beginnt man z.B. bei der Summe innen in den Klammern und baut zuerst ein NICHT a. Dessen Ausgang kommt zusammen mit b in einen UND-Baustein. Aber sehen sie selbst:



Etwas ungewöhnlich ist die Darstellung der Eingänge a und b als senkrechte Linien. Hier stellen sie sich bitte zwei Schalter vor, mit denen man diese senkrechten Leitungen "ein - und ausschalten" kann.

Der obere Teil der Schaltung (und damit die Logikgleichung für die Summe) läßt sich auch als XOR darstellen. XOR ist das Exklusiv-Oder, das wir auch umgangssprachlich immer meinen, wenn es "entweder - oder " heißt. In der Logikgleichung ist das entsprechende Symbol ein Oder-Symbol mit einem Punkt darüber und das DIN-Schaltsymbol ist mit "=1" sehr einleuchtend. Wir verwenden hier auch dann die gängigen Symbole für Summe Σ und den Übertrag (Carry) $\mathbf{C_n}$

Die Schaltung sieht dann etwas einfacher aus.

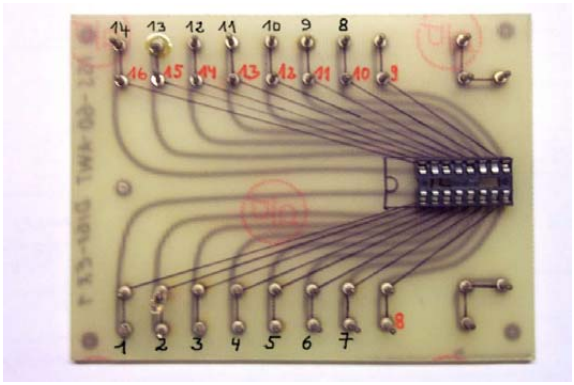


© Hans-Georg Beckmann 2002

Damit ist die Maschine auf dem Blatt Papier fertiggestellt und sie funktioniert natürlich asusgezeichnet. Im letzten Schritt soll nun diese Maschine tatsächlich gebaut weden.
Begeben sie sich also in den Physikraum oder in den privaten Bastelkeller, in dem es jetzt weitergeht.

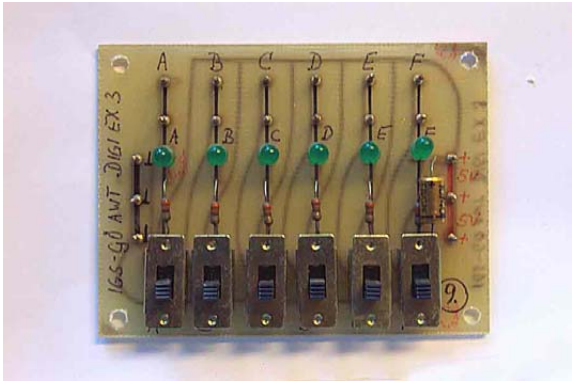
Hardware zum Aufbau digitaler Schaltungen

Wir werden alle Schaltungen mit Standard ICs realisieren. Das sind TTL-Bausteine (Transistor-Transitor-Logik), die in fast allen elektrischen und elektronischen Maschinen eingebaut sind. Die Bausteine stammen aus der 74er-Serie. Die heißt so, weil die Firma Texas Instruments den Bausteinen Nummern gegeben hat die mit 7400 beginnen, dann kommen 7401, 7402 u.s.w. Es gibt diese ICs in verschiedenen Bauformen, die man an einem Buchstabenkürzel zwischen der 74 und der nachfolgenden Zahl erkennen kann. 74N00 oder 74LS00 oder 74H00 sind beispielsweise verschiedene Ausgaben mit jeweils der gleichen Funktion. Weiteres zu diesen Bausteinen finden sie in dem kleinen Anhang zu diesem Kapitel.



Schaltbrett mit IC-Fassung (rechts) und Kontaktstiften (je zwei für jedes IC-Beinchen). Von Schülern gebaut.

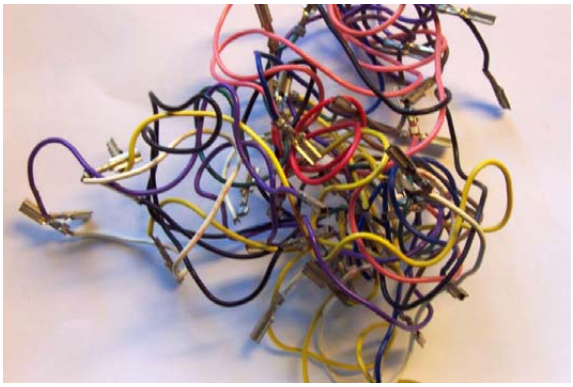
Schaltbrett mitLeuchtdioden zur Darstellung von Ausgabegrößen.



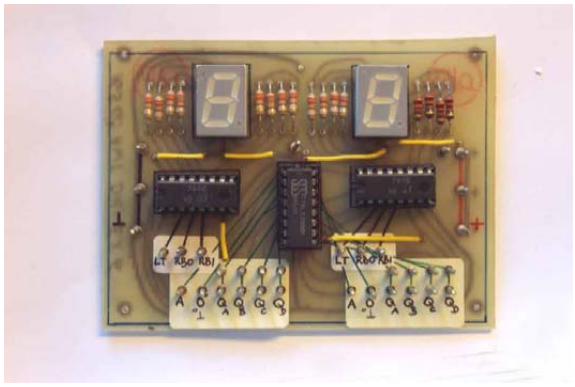
Schaltbrett mit 6 Eingabeschaltern undf Kontroll-LEDs

Kleines im Unterricht gebautes Netzteil.

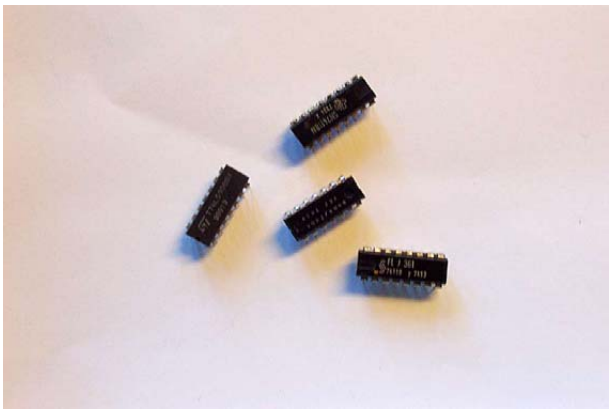
© Hans-Georg Beckmann 2002



Kabelmaterial mit Steckschuhen (selbst gebaut)



Auch komplexere Bausteine sind möglich. Ein Zähler mit Siebensegmentanzeigen als Ausgabe

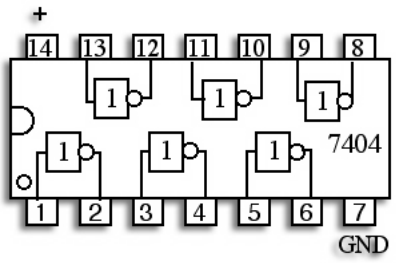


Man braucht nun auch Fassungen, in die man die ICs stecken kann. Es gibt für schulische Zwecke zwei Möglichkeiten, die den Schuletat nicht zu sehr strapazieren: Selbst bauen oder Steckbretter z.B. der Firma Hirschmann verwenden.
Ein Netzgerät, das 5V Gleichspannung liefert ist sicher in allen Physikräumen vorhanden. Man kann aber auch kleine Gleichrichter selbst bauen, die dann mit 10V bis 12V Wechselspannung betrieben werden.
Schalter und Leuchtdioden sind für die Ein - und Ausgabe der Werte notwendig.
Viele der Materialien können selbst gebaut werden. Die Zeit wird man aber im normalen Unterricht nicht haben. Daher muss man dann auf alternative Lösungen zurückgreifen (siehe Materialanhang).
In jedem Fall braucht man eine große Auswahl an ICs der 74er Serie. Die kosten aber in den meisten Fällen nur wenige Cent und belasten den Etat daher kaum.

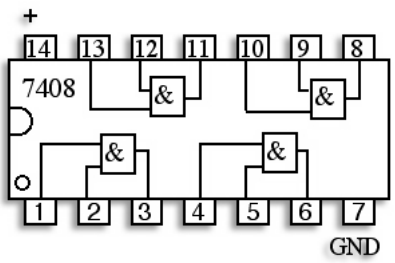
Das Innenleben der Standard ICs

Für unseren Halbaddierer brauchen wir UND-, ODER- und NICHT-Bausteine. Auf den XOR-Baustein verzichten wir hier noch.
Der 7404 enthält NICHT, der 7408 enthält UND und der Baustein 7432 enthält ODER.
Die Spannungsversorgung liegt bei allen drei Bausteinen an den Beinchen 7 und 14.
Im Inneren sind diese 14-beinigen ICs wie folgt aufgebaut.

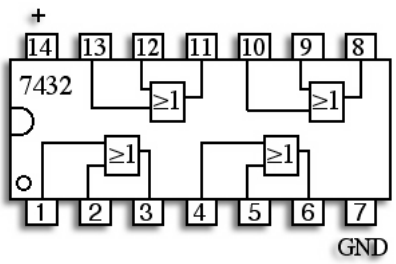
© Hans-Georg Beckmann 2002



7404
6 fach Inverter (NICHT)

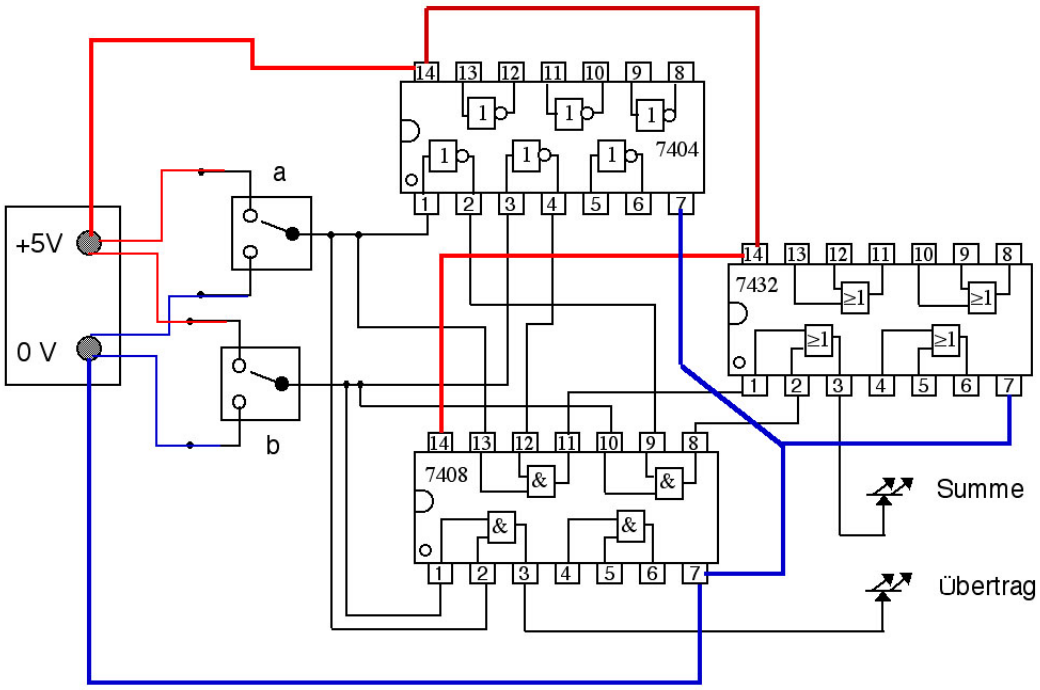


7408
4 fach UND



7432
4 fach ODER

Nun kann man die Schaltung für den Halbaddierer zusammenbauen. Als Ausgabe werden Leuchtdioden benutzt, zur Eingabe Schalter, die auf 5V bzw. 0 V geschaltet werden können.



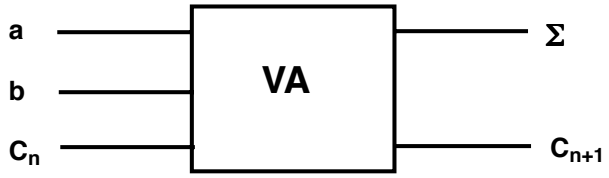
Das sieht im ersten Moment recht unübersichtlich aus. Probieren sie die Schaltung aufzubauen. Bei Fehlern überprüfen sie zuerst immer die Kabel. Wenn es nicht gleich klappt, nicht entmutigen lassen.

Schaltungsentwicklung am Beispiel des Volladdierers

In den Beispielrechnungen auf Seite 5 kann man sehen, dass der Halbaddierer nur die Addition in der Stelle ganz rechts korrekt durchführen kann. In den nachfolgenden Stellen braucht die

© Hans-Georg Beckmann 2002

Schaltung einen dritten Eingang, da möglicherweise der Übertrag der vorangegangenen Stelle mit verrechnet werden muss. Der Volladdierer muss also einen Übertragseingang C_n haben und einen Ausgang für den Übertrag in die nächste Stelle, der hier mit C_{n+1} bezeichnet ist.



Schaltungen, wie den Volladdierer kann man leicht in drei Schritten entwickeln.

Schritt 1: Die Darstellung als Tabelle

In einer Tabelle werden alle Eingangsgrößen mit allen möglichen Wertekombinationen erfasst und jeweils der gewünschte Wert des Ausgangs notiert. Dabei ist es sinnvoll, die Bitkombinationen der Eingangsgrößen der richtigen Reihenfolge nach aufzulisten.

Bei drei Eingängen hat man 2^3 Zeilen in der Tabelle.

a	b	C_n	Σ	C_{n+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Schritt 2: Die Logikgleichungen

Aus der Tabelle werden die Logikgleichungen gewonnen, indem man für Summe und Übertrag die Zeilen betrachtet, bei denen eine "1" als Ergebnis steht. Man kann für diese Zeilen den entsprechenden Teil der Logikgleichung direkt aus der Tabelle ablesen. Die einzelnen Teile werden dann mit "ODER" verknüpft.

$$\Sigma = (\bar{a} \wedge \bar{b} \wedge c_n) \vee (\bar{a} \wedge b \wedge \bar{c_n}) \vee (a \wedge \bar{b} \wedge \bar{c_n}) \vee (a \wedge b \wedge c_n)$$

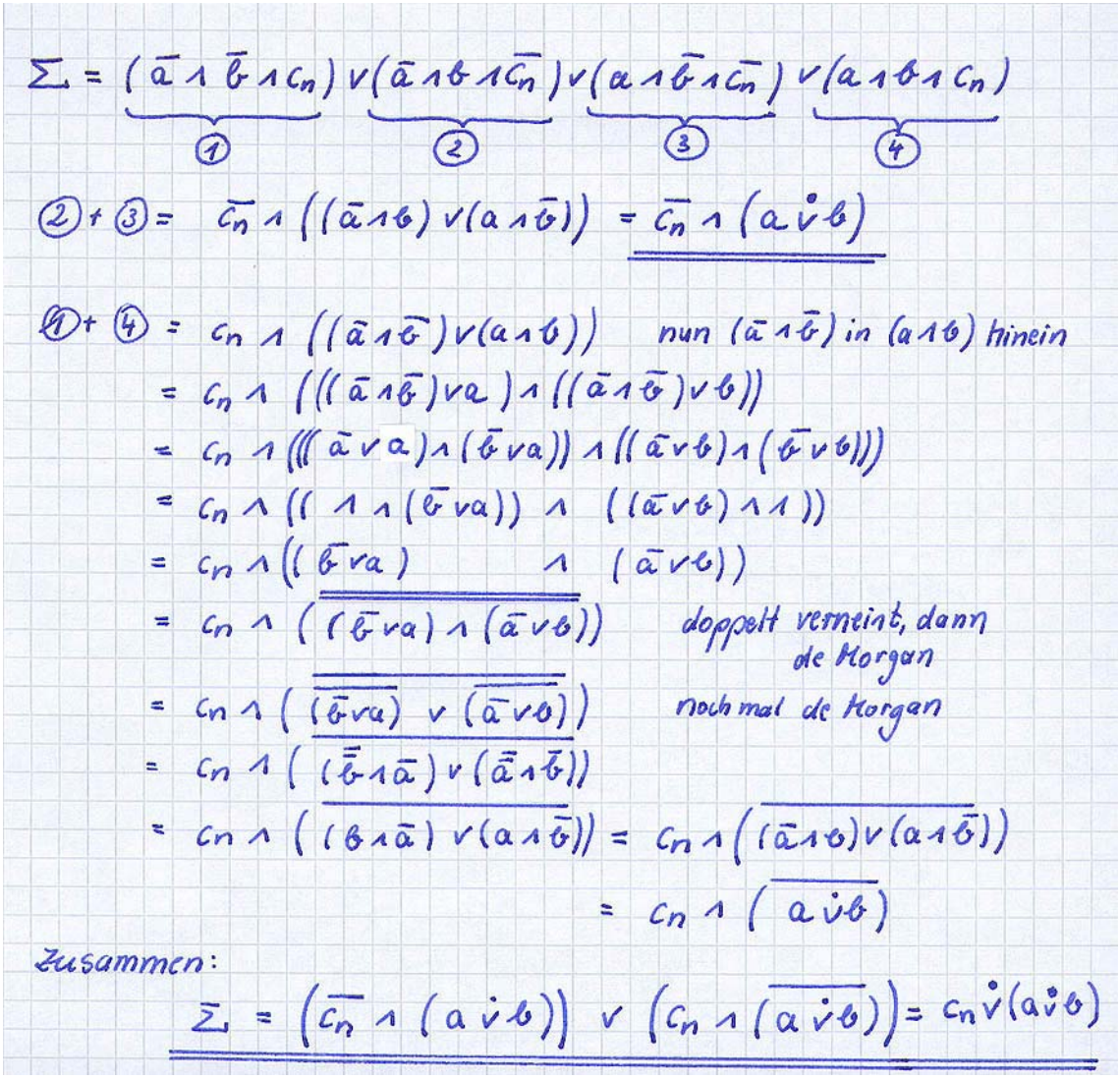
Zeile 2 Zeile 3 Zeile 5 Zeile 8

$$C_{n+1} = (\bar{a} \wedge b \wedge c_n) \vee (a \wedge \bar{b} \wedge c_n) \vee (a \wedge b \wedge \bar{c_n}) \vee (a \wedge b \wedge c_n)$$

Zeile 4 Zeile 6 Zeile 7 Zeile 8

© Hans-Georg Beckmann 2002

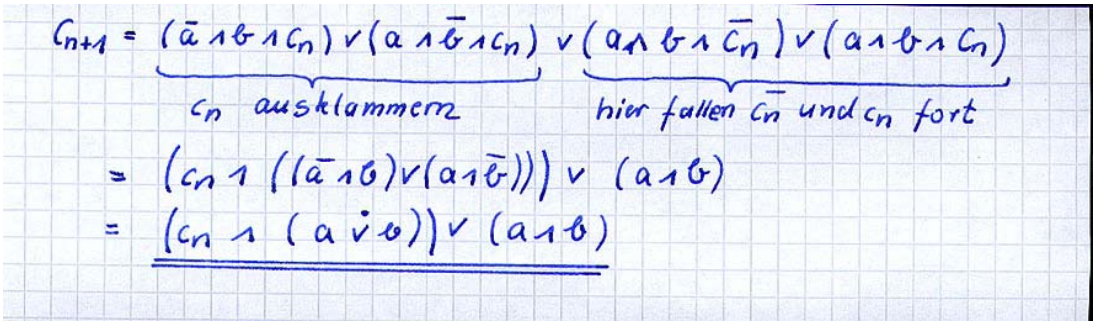
Hier kommt nun der schwierige Teil. Die Gleichungen sollen vereinfacht werden. Dazu kann man die Gesetze benutzen, die oben angegeben sind.
In der Gleichung für die Summe kann man zuerst ausklammern. Die nachfolgende Rechnung sollten sie schrittweise durcharbeiten.



Am Ende läßt sich die Summe als doppelt geschachtelte XOR-Verknüpfung darstellen.
Dabei entspricht die Klammer der Summe von a und b, wie wir sie aus unserem Halbaddierer kennen. Das Ergebnis wird dann noch mit c_n auf die gleiche Weise verknüpft.

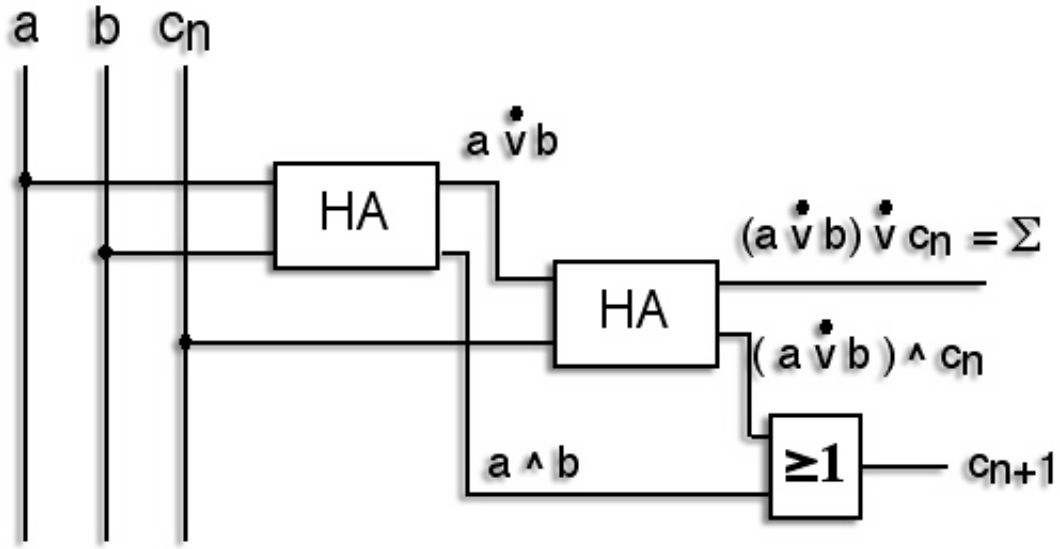
Hier bietet sich schon an, für die ermittlung der Summe zwei Halbaddierer zu verwenden. Bleibt die Frage, ob man den Übertragsteil unseres Halbaddierers auch mit verwenden kann, um in diesem Volladdierer den Übertrag zu ermitteln.
Betrachten sie die Gleichung für den Übertrag:

© Hans-Georg Beckmann 2002

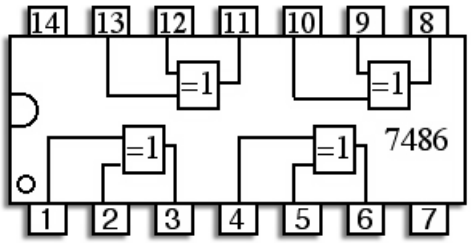


Es klappt tatsächlich. C_n und die Summe aus dem ersten Halbaddierer können den Übertrag erzeugen, oder aber die Summe aus a und b.

Schritt 3: Die Schaltung bauen



Dabei sind die Halbaddierer so aufgebaut, wie oben gezeigt. Wenn man nun noch einen entsprechenden Baustein für XOR hätte.... Man hat ! Es ist der 7486.



Wie nun schon nicht mehr anders zu erwarten, gibt es natürlich auch einen TTL-Baustein, der gleich einen ganzen Volladdierer enthält. Es ist der Baustein 7480, der kaum noch im Elektronikhandel zu bekommen ist.
Dieser Baustein hat einige schaltungstechnische Besonderheiten, die aber für unsere Zwecke nicht wichtig sind. Genauso geeignet ist der Baustein 7482, der gleich 2 Volladdierer enthält.

© Hans-Georg Beckmann 2002

1-Bit-Volladdierer (7480)

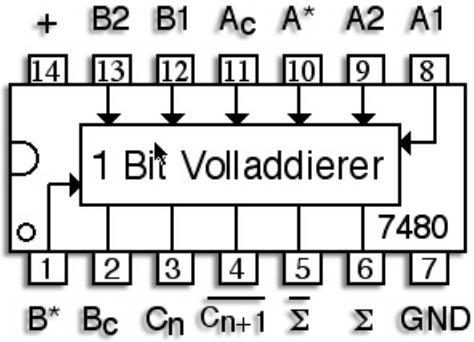
Der Übertrag c_{n+1} steht nur invertiert zur Verfügung.

Die Summanden können an A1,B1 oder A2,B2 oder invertiert an A*,B* angelegt werden.

Werden A* und B* verwendet müssen A1,A2,B1 und B2 auf Masse (0V) gelegt werden. Werden A1,B1 oder A2,B2 verwendet müssen A* und B* unbeschaltet beliben (offen). Intern arbeitet der Baustein nur mit Eingängen A und B, die wie folgt errechnet werden:

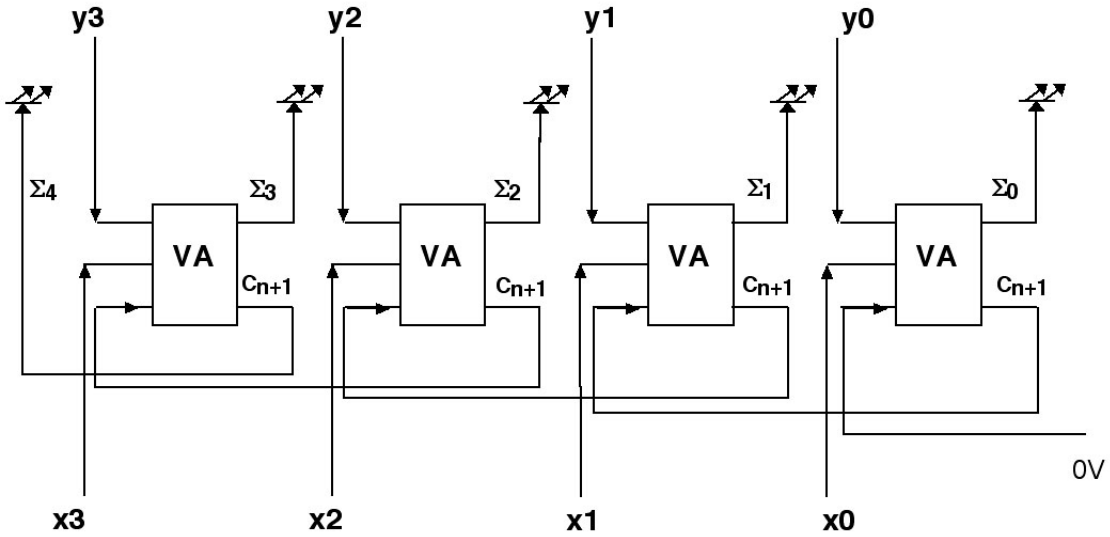
$A = \overline{A^* \wedge A_c}$ und $B = \overline{B^* \wedge B_c}$ wobei wiederum

$A^* = \overline{A1 \wedge A2}$ und $B^* = \overline{B1 \wedge B2}$ ist.



Schritt 2 bei diesem Beispiel war recht kompliziert. Natürlich ist es auch möglich, den Aufbau der Schaltung ohne vorherige Bearbeitung der Gleichung durchzuführen. Lassen sie sich durch dieses Beispiel nicht zu sehr erschrecken, es wird wieder übersichtlicher.

Bleibt am Ende nur noch der letzte Schritt, den kompletten Addierer zu bauen, der zwei 4 -Bit - Zahlen addieren kann. Wir werden dazu entweder einen Halbaddierer und drei Volladdierer brauchen oder aber mit vier Volladdieren arbeiten, wobei einmal (für die Rechenstelle ganz rechts) der Übertragseingang fest auf Null gelegt wird.



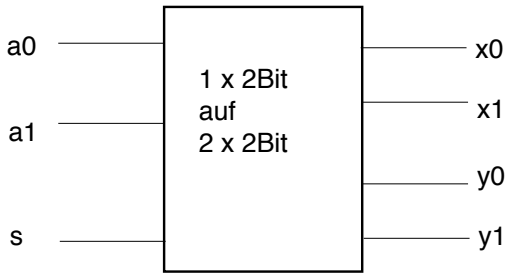
x0 und y0 sind die niedrigsten Bits der beiden Dualzahlen. Das Ergebnis kann eventuell 5 Stellen haben (Σ0 bis Σ4).

Es gibt einen fertigen Baustein (7483) der zwei 4-Bit-Zahlen addieren kann.

© Hans-Georg Beckmann 2002

Beispiel 1: Eine Verteilerschaltung

Eine Schaltung habe ein Paar Eingangsleitungen a0,a1 und zwei Paar Ausgangsleitungen x0,x1 und y0, y1. Eine Steuerleitung **s** ist weiterer Eingang. Wenn **s** Auf logisch "0" steht, werden die Eingänge a0, a1 auf x0, x1 geschaltet, während y0 und y1 in jedem Falle auf "0" bleiben. Ist s=1, werden die Werte von a0, a1 auf y0, y1 durchgeschaltet und x0, x1 bleiben auf Null. Die Schaltung verteilt also eine Datenquelle auf zwei Ziele .



Schritt 1: Die Tabelle

Die Tabelle wird hier zweckmäßigerweise so dargestellt, dass zuerst die 4 Zeilen für den Fall s=0 aufgeführt sind und dann die vier Zeilen für den Fall s=1.

a0	a1	s	x0	x1	y0	y1
0	0	0	0	0	0	0
0	1	0	0	1	0	0
1	0	0	1	0	0	0
1	1	0	1	1	0	0
0	0	1	0	0	0	0
0	1	1	0	0	0	1
1	0	1	0	0	1	0
1	1	1	0	0	1	1

Schritt 2: Die Logikgleichungen

Für 4 mögliche Ausgänge muss es auch 4 Gleichungen geben:

$$x0 = (a0 \wedge \overline{a1} \wedge \overline{s}) \vee (a0 \wedge a1 \wedge \overline{s}) \quad \text{und} \quad x1 = (\overline{a0} \wedge a1 \wedge \overline{s}) \vee (a0 \wedge a1 \wedge \overline{s})$$

$$y0 = (a0 \wedge \overline{a1} \wedge s) \vee (a0 \wedge a1 \wedge s) \quad \text{und} \quad y1 = (\overline{a0} \wedge a1 \wedge s) \vee (a0 \wedge a1 \wedge s)$$

Es ergibt sich sofort eine Vereinfachung, wenn man bedenkt, dass Klammern die sich nur in einem Wert unterscheiden , $(a \wedge \overline{b} \wedge c) \vee (a \wedge b \wedge c) = (a \wedge b)$ gilt:

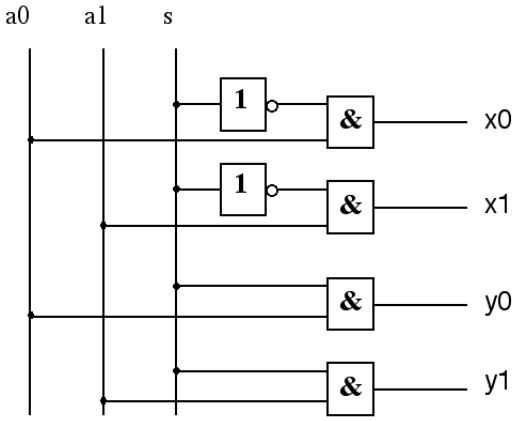
Damit erhält man folgende Ergebnisse:

$$x0 = (a0 \wedge \overline{s}) \quad \text{und} \quad x1 = (a1 \wedge \overline{s}) \quad \text{und} \quad y0 = (a0 \wedge s) \quad \text{und} \quad y1 = (a1 \wedge s)$$

Das hätte man auch direkt aus der Tabelle hinschreiben können, denn wie schon die Beschreibung der Schaltung angibt, kann z.N: x0 nur 1 sein, wenn **s** auf 1 liegt und a0 auf 1.

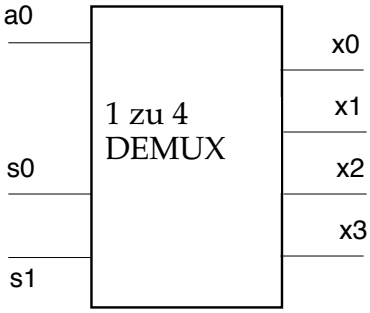
© Hans-Georg Beckmann 2002

Schritt 3: Die Schaltung



Beispiel 2: Demultiplexer

Ein Demultiplexer schaltet eine Eingangsleitung a0 auf einen der Ausgänge x0,x1,x2,x3 durch, wobei die nicht ausgewählten Ausgänge auf Null stehen sollen. Zur Auswahl der Ausgangsleitung werden zwei Steuerleitungen (Adressleitungen) s0 und s1 verwendet. Entwicklen sie die Schaltung dieses "1 zu 4 Demultiplexers".



Die Tabelle sei wie nebenstehend vorgegeben:

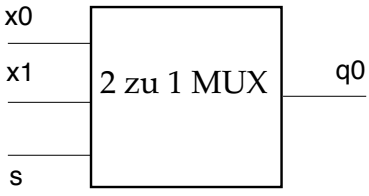
a0	s0	s1	x0	x1	x2	x3
0	0	0	0	0	0	0
1	0	0	1	0	0	0
0	0	1	0	0	0	0
1	0	1	0	1	0	0
0	1	0	0	0	0	0
1	1	0	0	0	1	0
0	1	1	0	0	0	0
1	1	1	0	0	0	1

Stellen sie die Logikgleichungen zusammen. Zeichnen sie eine Schaltung.

Bauen sie aus dieser Schaltung einen (1x 2 Bit) zu (4 x 2 Bit) Demultiplexer, der mithilfe von Steuerleitungen s0,s1 zwei Bits a0 und a1 auf eines der vier Ausgangspaare x0,y0 oder x2,y1 oder x2,y2 oder x3,y3 durchschaltet.
Vergleichen sie auch mit der Schaltung auf Seite 18.

Beispiel 3: Multiplexer

Ein Multiplexer schaltet mithilfe von Steuerleitungen (Adressleitungen) eine von mehreren Eingangsleitungen auf einen Ausgang durch. Entwickeln sie zuerst einen 2x1-Bit zu

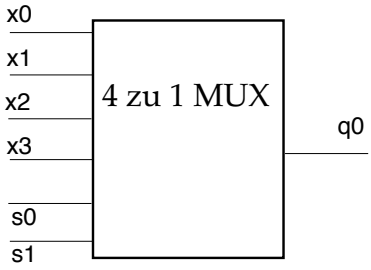


© Hans-Georg Beckmann 2002

1x1 Bit Multiplexer, der den Eingang x0 auf den Ausgang q0 durchschaltet, wenn die Steuerleitung s auf 0 steht und der den Eingang x1 auf q durchschaltet, wenn s auf 1 steht.
Bauen sie daraus unter Verwendung des Symbols auf Seite 19 unten einen 2 x 2 Bit zu 1 x 2 Bit Multiplexer und dann einen 4 x 2 Bit zu 1 x 2 Bit Multiplexer. Dabei muss keine neue Tabelle erstellt werden und es müssen auch keine neuen Logikgleichungen aufgestellt werden.

Beispiel 4:

Ein 4 x 1 Bit zu 1 x 1 Bit Mulitplexer mit zwei Adressleitungen s0 und s1.
Entwicklen sie die Schaltung !



Beispiel 5: Schwellwertschaltung

Eine Schwellwertschaltung hat drei Eingänge x0,x1 und x2. Der Ausgang q geht auf "1", wenn mindestens zwei der Eingangsleitungen auf "1" stehen. Entwicklen sie die Schaltung !

Beispiel 6: Siebensegmentanzeige

Eine Siebensegmentanzeige mit BCD Eingängen

Vorbemerkungen

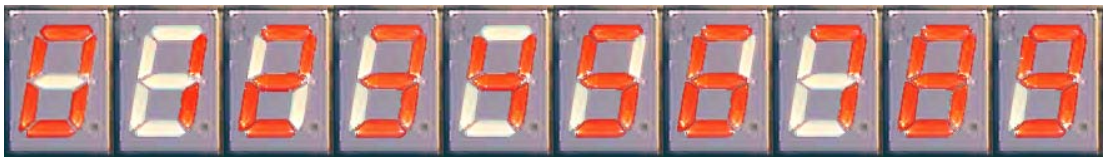
BCD- Zahlen sind Binär - Codierte - Dezimalzahlen. Sie sehen fast so aus , wie die Dualzahlen sind aber nichts anderes, als die Darstellung jeder Ziffer eine Zahl aus dem Dezimalsystem durch eine vierstellige Dualzahl. z.B. hätte die Zahl 2357 die BCD-Darstellung 0010 0011 0101 0111 wenn jede Ziffer als 4 Bit Dualzahl dargestellt wird.

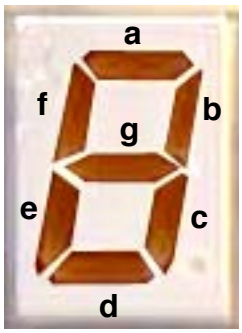
Zahl	Ziffern		Dualdarstellung Ziffer 1					Dualdarstellung Ziffer 0			
0		0						0	0	0	0
1		1						0	0	0	1
2		2						0	0	1	0
3		3						0	0	1	1
4		4						0	1	0	0
5		5						0	1	0	1
6		6						0	1	1	0
7		7						0	1	1	1
8		8						1	0	0	0
9		9						1	0	0	1
10	1	0		0	0	0	1	0	0	0	0
11	1	1		0	0	0	1	0	0	0	1
12	1	2		0	0	0	1	0	0	1	0
13	1	3		0	0	0	1	0	0	1	1
14	1	4		0	0	0	1	0	1	0	0
15	1	5		0	0	0	1	0	1	0	1

Eine Siebensegmentanzeige kann aus BCD codierten Ziffern, wieder sichtbare Dezimalziffern machen, in dem sie verschiedene Segmente aufleuchten läßt . Eine Schaltung soll 4 Eingänge haben, die mit x0,x1,x2 und x3 bezeichnet seien. Die Schaltung hat 7 Ausgänge (für

© Hans-Georg Beckmann 2002

jedes Segment einen). Steht ein Ausgang auf "1" bedeutet das, dass das entsprechende Segement leuchtet.





Aufbau der Anzeige:

Tabelle:

Zahl	x3	x2	x1	x0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	1	1	1

Beispielgleichung für das Segment e:

$e = (\overline{x0} \wedge \overline{x1} \wedge \overline{x2} \wedge \overline{x3})$

$v\ (\overline{x0} \wedge x1 \wedge \overline{x2} \wedge \overline{x3})\ v(\overline{x0} \wedge x1 \wedge x2 \wedge \overline{x3})$

$v(\overline{x0} \wedge \overline{x1} \wedge \overline{x2} \wedge x3)\ v(x0 \wedge \overline{x1} \wedge \overline{x2} \wedge x3)$

Klammern zusammenfassen:

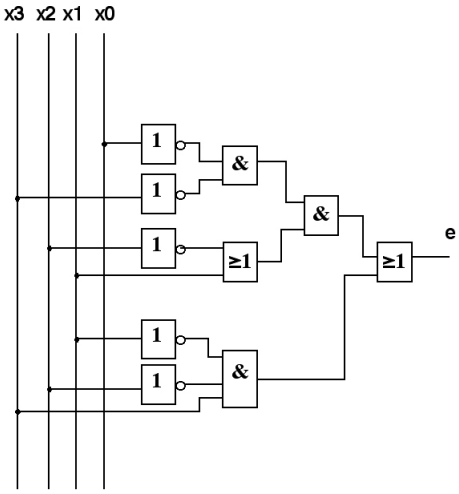
$e = (\overline{x0} \wedge \overline{x2} \wedge \overline{x3})\ v\ (\overline{x0} \wedge x1 \wedge x2 \wedge \overline{x3})\ v\ (\overline{x1} \wedge \overline{x2} \wedge x3)$

$e = ((\overline{x0} \wedge \overline{x3}) \wedge ((\overline{x2}) v(x1 \wedge x2))) v\ (\overline{x1} \wedge \overline{x2} \wedge x3)$

$e = ((\overline{x0} \wedge \overline{x3}) \wedge ((\overline{x2} v x1) \wedge (\overline{x2} v x2))) v\ (\overline{x1} \wedge \overline{x2} \wedge x3)$

$e = ((\overline{x0} \wedge \overline{x3}) \wedge (\overline{x2} v x1)) v\ (\overline{x1} \wedge \overline{x2} \wedge x3)$

Man kann sicher noch weiter umformen, wir verzichten hier darauf und werden später darauf zurückkommen.
Die nebenstehende Schaltung ist noch annehmbar. Allerdings wird es schon schwierg werden,wenn man z.B. das Segment b betrachtet. Dort warten 8 Zeilen mit eine "1" auf die Logikgleichungen. Das muss auch einfacher gehen !



© Hans-Georg Beckmann 2002

Vereinfachung von Logikgleichungen

1. Verfahren nach Karnaugh-Veitch

Bei dem folgenden Verfahren liegen zwei Regeln zugrunde:

I: Wenn Klammmer in Logikgleichungen sich nur in *einem* Wert unterscheiden, dann kann dieser Wert wegfallen:

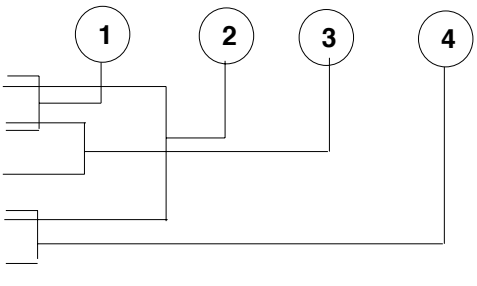
$$(\bar{a} \wedge x) \vee (a \wedge x) = x \quad \text{bzw.} \quad (\bar{a} \vee x) \wedge (a \vee x) = x$$

II: Man darf einen Wert in einer Logikgleichung auch verdoppeln bzw. mehrfach verwenden, denn es gilt offensichtlich:

$$(x \wedge x) = x \quad \text{bzw.} \quad (x \vee x) = x$$

Um diese Vereinfachungen zu erreichen gibt es ein graphisches Verfahren auf das wir hier nicht eingehen, da für uns eine tabellarische Darstellung vollkommen ausreicht.
Als Beispiel dient noch einmal das Segment *e* aus der Siebensegmentschaltung.
Wir schreiben nur die Zeilen auf, in denen das Ergebnis "1" ist:

Zahl	x3	x2	x1	x0	e
0	**0**	**0**	**0**	**0**	1
2	**0**	**0**	**1**	**0**	1
6	**0**	**1**	**1**	**0**	1
8	**1**	**0**	**0**	**0**	1
9	**1**	**0**	**0**	**1**	1



(1) $= (\bar{x3} \vee \bar{x2} \vee \bar{x0})$ Die Zeilen 1 und 2 unterscheiden sich nur in x1. x1 kann entfallen .

(2) $= (\bar{x2} \vee \bar{x1} \vee \bar{x0})$ Die Zeilen 1 und 4 unterscheiden sich nur in x3. x3 kann entfallen.

(3) $= (\bar{x3} \vee x1 \vee \bar{x0})$ Die Zeilen 2 und 3 unterscheiden sich nur in x2. x2 kann entfallen.

(4) $= (x3 \vee \bar{x2} \vee \bar{x1})$ Die Zeilen 4 und 5 unterscheiden sich nur in x0. x0 kann entfallen.

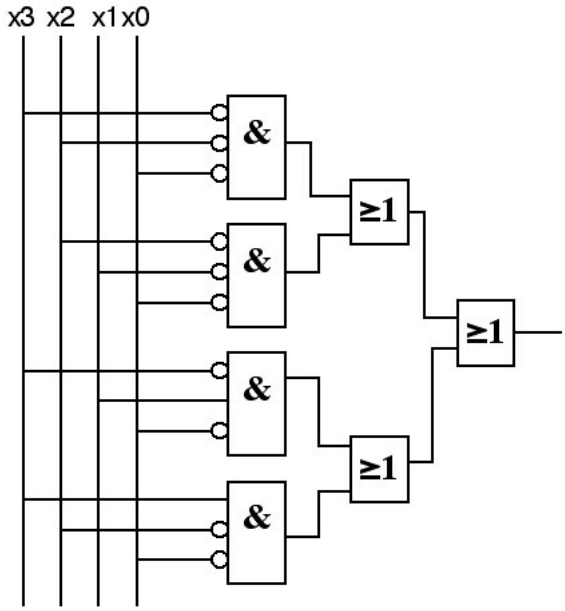
Da alle Zeilen (Klammern) verbraucht wurden ist das Ergebnis nun:

$$e = (\bar{x3} \vee \bar{x2} \vee \bar{x0}) \wedge (\bar{x2} \vee \bar{x1} \vee \bar{x0}) \wedge (\bar{x3} \vee x1 \vee \bar{x0}) \wedge (x3 \vee \bar{x2} \vee \bar{x1})$$

Wir würden nun garnicht mehr versuchen, diese Klammern weiter zusammenzufassen, obwohl das sicher möglich wäre. Eine übersichtliche Schaltung ergäbe sich , wenn man 3-fach-UND verwendet (die gibt es auch als TTL-Bausteine) und in der Schaltung nicht NICHT- Bausteine, die

© Hans-Georg Beckmann 2002

vor den Eingängen der UND-Bausteine plaziert werden müssen durch ein einfaches Negationssymbol (den NICHT-Kringel) dargestellt.



2. Die UND-Form der Logikgleichung

Wir haben bis jetzt immer nach den Zeilen in den Tabelle geschaut, bei denen als Ergebnis eine "1" auftritt, Wir werden jetzt nach den Zeilen suchen, bei denen eine "0" auftaucht. Das sind in einigen Fällen sehr viel weniger Zeilen !

Vorüberlegungen an einem einfachen Beispiel:

Man habe nebenstehende Tabelle:

a	b	Q
0	0	1
0	1	0
1	0	1
1	1	1

Hier würden wir nach unserem bisherigen Verfahren als Gleichung finden:

$$Q = (\overline{a} \wedge \overline{b}) \vee (a \wedge \overline{b}) \vee (a \wedge b) = (\overline{b} \wedge (\overline{a} \vee a)) \vee (a \wedge b) = \overline{b} \vee (a \wedge b) = \overline{b} \vee a$$

Wir können aber auch nach der einzigen Zeile schauen, in der Q nicht "1" ist und erhalten:

$$\overline{Q} = (\overline{a} \wedge b)$$

Wir negieren die ganze Gleichung und erhalten:

$$\overline{Q} = Q = \overline{(\overline{a} \wedge b)} = (\overline{a} \vee \overline{b}) = (a \vee \overline{b})$$

Man kann das Ergebnis auch direkt hinschreiben, wenn man sich an folgende Regeln hält:
Suche die Zeilen mit "0" im Ergebnis. Schreibe die Eingangsgrößen invertiert auf, verknüpfe innehalb der Klammern mit "ODER" und verknüpfe die Klammern mit "UND".

Dieses ist die sogenannte UND-Form der Logikgleichungen, während das bisherige Verfahren die ODER-Form war. Hat man in einer Tabelle weniger "1" als "0" dann nimmt man die ODER. Hat man weniger "0" als "1" , dann ist die UND-Form angebracht.

Schaut man sich das noch einmal am Beispiel der XOR-Tabelle an, dann kann man beide Lösungen nebeneinander sehen und vergleichen:

© Hans-Georg Beckmann 2002

$Q = (\overline{a} \wedge b) \vee (a \wedge \overline{b})$ (ODER - Form)

$Q = (a \vee \overline{b}) \wedge (\overline{a} \vee b)$ (UND - Form)

a	b	Q
0	0	0
0	1	1
1	0	1
1	1	0

So recht glauben mag man das nicht, obwohl es eine ähnliche Umformung schon beim Volladdierer gab.
Für die Zeilen 1 und 3 aus der Tabelle gilt:

$$\overline{Q} = (\overline{a} \wedge \overline{b}) \vee (a \wedge b) = (a \vee (\overline{a} \wedge \overline{b})) \wedge (b \vee (\overline{a} \wedge \overline{b}))$$
$$= ((a \vee \overline{a}) \wedge (a \vee \overline{b})) \wedge ((b \vee \overline{a}) \wedge (b \vee \overline{b}))$$
$$= (a \vee \overline{b}) \wedge (b \vee \overline{a}) \text{ nun alles negieren:}$$
$$\overline{\overline{Q}} = Q = \overline{(a \vee \overline{b}) \wedge (b \vee \overline{a})} = (\overline{a} \wedge b) \vee (a \wedge \overline{b})$$

In der letzten Zeile wurden die deMorgangschen Gesetze verwendet und man erhält am Ende die ODER-Form.

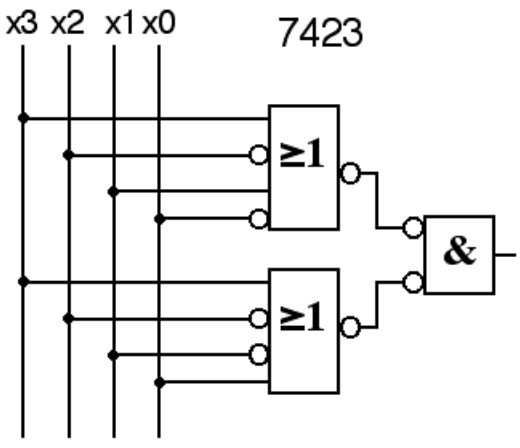
UND-Form am Beispiel des Segments b

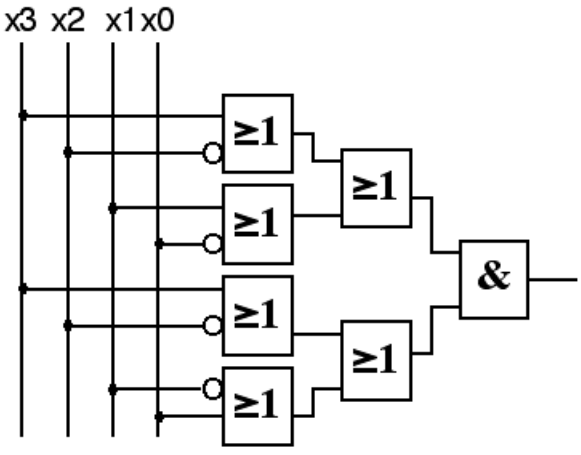
In der Tabelle sieht man 7 Zeilen mit einer "1" als Ergebnis, aber nur 2 Zeilen mit einer "0". Damit bietet sich die UND-Form an: Auf weitere Zusammenfassung werden wir auch hier verzichten.

$$b = (x3 \vee \overline{x2} \vee x1 \vee \overline{x0}) \wedge (x3 \vee \overline{x2} \vee \overline{x1} \vee x0)$$

Eine Schaltung ist mit 4-fach NOR-Bausteinen möglich, deren Ausgänge man nur invertieren muss. Man kann aber auch mit üblichen ODER-Buasteinen arbeiten.

Zahl	x3	x2	x1	x0	**b**
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	**0**	**1**	**0**	**1**	**0**
6	**0**	**1**	**1**	**0**	**0**
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1



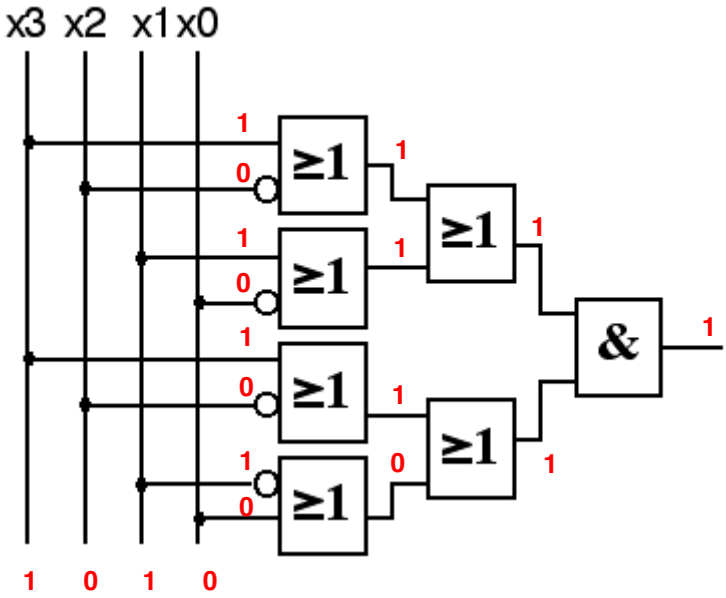


Schaltung mit 4-fach NOR (TTL 7423) oder alternativ ODER (TTL 7432)

© Hans-Georg Beckmann 2002

Weitere Aufgaben zur Siebensegmentanzeige

Offensichtlich haben wir in der Tabelle auf Seite 21 nicht alle Kombinationen der Eingangsgrößen beachtet. Unsere Tabelle hätte mehr Zeilen haben müssen. Auch wenn wir davon ausgehen, dass unsere Schaltung insgesamt so sein soll, dass keine unzulässigen Kombinationen an den Eingängen auftreten, bleibt die Frage, was passiert, wenn soetwas doch vorkommt. Dazu betrachtet man sich die Schaltung und geht in ihr die einzelnen Bausteine durch. Betrachten wir das Segment b für die unzulässige Eingabe "1010". An die Leitungen wird jeweils geschrieben, was der jeweile Baustein als Ergebnis produziert.



Offensichtlich leuchtet das Segment b auf !
Bei einer Siebensegmentanzeige mag man noch sicher sein, dass die vorgeschalteten Bausteine schon korrekte BCD-Zahlen liefern. Stellen sie sich aber so ein unklares Verhalten bei einer Alarmanlage vor . Man muss also im Zweifelsfall auch die Eingangskombinationen in der Tabelle erfassen und mit Ergebnissen versehen, die man für den eigentlichen Zweck der Schaltung nicht braucht.
Bei der Siebensegmentenzeige wäre es denkbar, dass für alle nicht zulässigen BCD-Zahlen alle Segmente dunkel bleiben oder aber ein "E" für Error aufleuchtet. Die Logikgleichungen für die einzelnen Segmente werden dann natürlich noch unübersichtlicher. Zum Glück gibt es fertige Bausteine (das wundert sie doch nicht mehr !), das sind die TTLs 7446,7447 und 7448.

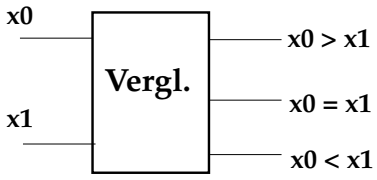
© Hans-Georg Beckmann 2002

Weitere Übungsaufgaben

Beispielaufgabe 7:

Entwickle eine Schaltung mit 4 Eingängen x3,x2,x1 und x0, die an ihrem einzigen Ausgang Q genaus dann eine "1" zeigt, wenn die Anzahl der "1" an den Eingängen ungerade ist.

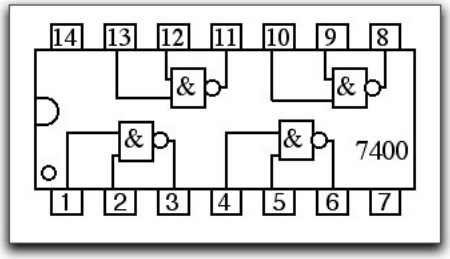
Beispielaufgabe 8:



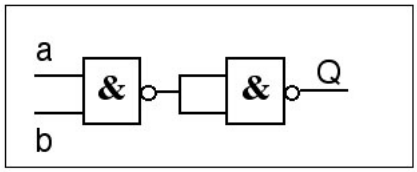
Entwickle eine 2 x 1 Bit Vergleicherschaltung. Die drei Augänge gehen jeweils auf "1" wenn die entsprechende Vergleichsbedingung erfüllt ist:
Entwickle daraus einen 2 x 4 Bit Vergleicher, der zwei 4 Bitzahlen vergleichen kann.

Beispielaufgabe 9:

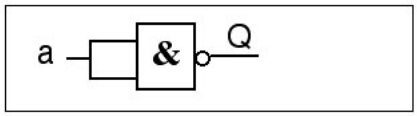
Baue die folgenden drei Schaltungen auf oder untersuche sie "auf dem Papier". Beim Aufbau muss der TTL 7400 verwendet werden, der 4 NAND-Bausteine enthält. Was leisten die drei Schaltungen ?



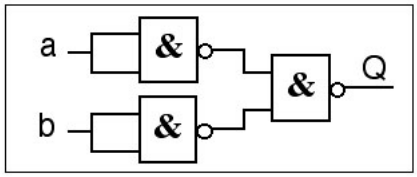
I



II



III



Beispielaufgabe 10:

Eine Schaltung hat 4 Eingänge x3, x2, x1 und x0. Die Schaltung zählt die Anzahl der "1" , die an den Eingängen anliegt und gibt diese Anzahl als Dualzahl aus. Dazu braucht diese Schaltung zwei Ausgänge. Entwickle diese Schaltung.